PROGRAM PROTECTION NEWSLETTER
JUNE 1985

T IS A PROGRAM PROTECTION NEWSLETTER?

It is a monthly newsletter that will deal with program protection schemes of all types. Each month will cover 3 to 5 specific examples of different program protection schemes. Some will be very easy schemes, others may be very difficult. Some issues will cover word processors, data bases, spread sheets and games. Many different programs will be selected and each will be fully explained as to their type of protection.

Each program example used will be for illustrative purposes only. It is not the intention of this newsletter to promote or encourage illegal or unauthorized duplication of any copyrighted program. UNAUTHORIZED DUPLICATION OF A COPYRIGHTED PROGRAM IS ILLEGAL!!

WHY WOULD ANYBODY NEED A PROGRAM PROTECTION NEWSLETTER?

Many programs have a very sophisticated protection scheme and can not be used on some disk drives (MSD, 4040, 8050 etc.). Other programs use errors on the disk that can be very harmful to the disk drive. Many other people just prefer to use one disk for all their programs of one specific type (i.e. utilities).

According to federal copyright law you are allowed to make an ARCHIVAL copy of any program that you have purchased. You are also allowed to modify your program as long as the modifications are essential to the program's operation. Some protection schemes do not allow the program to be directly backed up. They may require the use of sophisticated copy programs or extensive knowledge of MACHINE LANGUAGE to understand. In either case the average user may not be able to obtain an archival copy of their valued programs. Another group of people just want to know more about their computer and its associated components. They want to know how programs are protected and how their machine operates. They want to know how to modify their own disks and how to protect their own programs.

Whatever reason you have for wanting to learn about program protection, this newsletter is for you!

PASS THIS COUPON ON TO YOUR FRIENDS WHO WISH TO LEARN MORE ABOUT PROGRAM PROTECTION SCHEMES. BETTER YET, USE IT TO RENEW YOUR SUBSCRIPTION FOR THE NEXT YEAR, AUG. 85 TO JULY 86.

SUBSCRIPTION PRICE $35.00 POST PAID IN U.S. AND CANADA
($45.00 FOR 1ST CLASS FOREIGN MAIL DELIVERY)

NAME ______________________________

ADDRESS ______________________________

CITY,ST,ZIP ______________________________

Renew by August for
continuous delivery.

CSM SOFTWARE INC.
P. O. BOX 563
CROWN POINT, IN 46307

* * * * * IMPORTANT NOTICE * * * * * * *

ANYONE USING AN AFTER MARKET POWER SUPPLY LABLED 'CPS-10' FOR THEIR COMMODORE COMPUTER: THIS POWER SUPPLY (available from HBH SALES) HAS A POTENTIALLY DANGEROUS DEFECT IN ITS DESIGN. IF YOU SUSPECT THAT YOU HAVE THIS POWER SUPPLY IMMEDIATELY DISCONNECT IT AND RETURN TO YOUR SUPPLIER FOR IMMEDIATE INSPECTION AND/OR REPLACEMENT. IF YOU BELONG TO A USER GROUP PLEASE PASS THIS INFORMATION ALONG TO YOUR MEMBERS. IT JUST MAY SAVE SOMEONE'S LIFE!!!

NO NEWS IS BAD NEWS DEPT. - ISEPIC

Last month we told you that an interesting new product had just been announced - ISEPIC. As we went to press last month, we had been promised an ISEPIC for evaluation, to arrive by overnight delivery. Well, here it is, five weeks and numerous phone calls later and still no ISEPIC. It's 'being shipped this week'. We'll see.

For all of you who called during the month for info on ISEPIC, we're sorry for the inconvenience. If and when we get an ISEPIC, we'll review it in the newsletter. Until then, we strongly suggest that you hold off purchasing one. Although the idea sounds promising, information we've obtained through the grapevine indicates that ISEPIC may not be all that it's cracked up to be. According to this information, it doesn't handle much beyond what can be done with copy utilities. It's also rumored to be easy to defeat so that even if it works on today's programs, it may be obsolete soon. We'll try to confirm or disprove these rumors as soon as we can, but there's not much we can do without the product! Stay tuned.

F-15 STRIKE EAGLE - UPDATE

Thanks to Joe Aufiero of Scotia, N.Y. for the following information.

There are apparently different versions of F-15 on the market. If the modifications given last month don't work for you, try this:

1. Load in "TITLE.BA" with HIMON. Line 301 contains a 'hidden' SYS 3416, which must be disabled by changing the $9E (SYS) to an $8F (REM). This byte is located at $0BEF in the version that we have, but yours may be different. Save the file back out with S "@0:TITLE.BA",08,0801,0FFB

2. Load in "F15.VID". Change bytes $7196-9D from $FF DF DD 90 71 F0 52 20 to $EA EA EA EA EA 4C EF 71. Again, these bytes may be at a different location in your version, so hunt for them. Save the file back out with S "@0:F15.VID",08,5100,7E00.

3. That's it.

The program is MOVIE MAKER, Copyright 1984, Interactive Picture Systems Inc.

TYPE OF PROTECTION: This program will check TRACK 2 for an error 23 and TRACK 3 for an error 27. That's right - TWO bangs for your buck.

HOW TO COPY: Copy the entire disk without errors and then put error 23's on all of TRACK 2 and error 27's on TRACK 3. Or use Omniclone, which will copy this program and place the errors for you.

HOW TO MAKE A WORKING COPY WITHOUT ERRORS ON THE DISK:

1. Make a copy of the original disk without errors (3 minute or file copy).

2. Load and execute LOMON with SYS 32768. Clean up the work space with F CC00 CFFF 99.

3. Load "MMO" from your original disk with L "MMO",08. Make the following changes using the M command. These values are inserted by the program if it passes the protection scheme.

```
CC04      90
CEF8-F9   FF FF
CF50-52   54 48 45
CF56      01
CFDB-DE   01 01 01 01
```

Save this file to your copy disk with S "@0:MO",08,CC00,D000.

4. You can now load and run the program as usual (load "*" or "MM") EXCEPT for the following. After the main menu appears the first time, you should select the PLAY option. Once PLAY has loaded (it's short), you can return to the MAIN MENU and choose any other option as usual. If you want to skip the opening credits, start with LOAD "MMO",8,1 and then SYS 52224. You should still choose PLAY the first time into the main menu.

5. YOU'RE DONE!

This program was done by the 'back-door' method. After starting the program normally from the original, we RESET the computer and examined memory. The "MMO" file contains the main menu and loads up at $CC00-CCFF. By comparing this area of memory with the original file, we found the differences given in step 3 above. After saving the modified "MMO" file back to a copy disk, we began examining the program more closely.

We found that each option module has its own protection check code, which is encrypted using EOR (exclusive-or). On the original disk, once a module has passed its check the first time, a value is saved which tells that module not to check again. The other modules, however, will still check the protection the first time they are loaded. On a copy disk with the modified "MMO" file, we found that the PLAY option works. Not only that, but it disables the protection for all the other modules. As long as we always choose the PLAY option first, we won't have to unprotect each option separately. Sometimes a little experimenting can save a lot of work.

The program is KOALAPAINTER (TM) by Koala Technologies Corp., Copyright 1983 by Audio Light, Inc.

TYPE OF PROTECTION: This program will check TRACK 33 for an ERROR 23.

HOW TO COPY: Use any copy program capable of placing an error 23.

HOW TO MAKE A WORKING COPY WITHOUT ERRORS ON THE DISK:

This is a program that will rattle your drive for absolutely no reason. The program is useless without the Koalapad hardware, and yet the programmer adds a needless protection scheme that will eventually throw your disk drive out of line.

If you examine the code from $39AC through $3A15, you will find that this program will read the error channel and store the values it finds at $C7D7 and $C7D8. It will check for the bad block first and return an error 23. Once the program passes this check, it will return to read a good block and store those values in the same location.

We will bypass the protection in the following manner:

1. Make a copy of the program without errors on the disk.

2. Load and run the program from the original disk. Once the program is running, RESET your computer.

3. Load and execute LOMON. Save out the code from $C000 through $C800 to your copy disk with S "@0:KPAINT1",08,C000,C800.

4. Now for the main body. Load the program from the original disk with L "KOALAPAINTER",08.

5. Using the M command at $0A81, change the $A0 to a $60 (RTS).

6. Save out the altered code with S "@0:KOALAPAINTER",08,0900,4600.

7. To utilize this program follow the procedure below:

   ```
   LOAD "KPAINT1",8,1
   LOAD "KOALAPAINTER",8,1
   SYS2304
   ```

   Your swap screen will come up black with this program. Just use your X-COLOR option to change the screen to white. You may wish to write a boot program that will load the two programs and SYS to the location given.

8. YOU'RE DONE!

The program is VIP TERMINAL (TM), Copyright 1983 by Softlaw Corp.

TYPE OF PROTECTION: The disk contains an ERROR 29 on TRACK 1 and 2, and an ERROR 23 on TRACK 18,6. The format code "A" on TRACK 18,0 / BYTE 02 has also been altered to prevent writing to the disk (see PPM Vol I for a discussion of the format code). The program will check for the presence of all of these errors.

HOW TO COPY: Use any copy program capable of placing these errors.

HOW TO COPY WITHOUT ERRORS ON THE DISK:

This program is stored on the disk in user files, which are loaded by a machine language loader. The loader is located on the disk on TRACK 19, SECTORS 0-6. We will make our modifications directly on the disk.

1. Make a copy of the disk without errors. Next, the format code on TRACK 18/0 must be corrected so you can write on the disk. You need to change BYTE 02 from an $01 to a $41 (DECIMAL 65, "A"). Since the disk cannot be written to by ordinary means yet, you must use the UN-WRITE PROTECTOR program given below to make this change.

2. With the format code changed, we will now make the other changes necessary for the proper execution of this program.

3. Using a track and sector editor, make the following changes:

TRK/SEC	BYTE LOC.	ORIGINAL	MODIFIED
19/0	7 & 19	32 (both)	30 (both)
19/1	123-125	20 27 1A	4C EC 18
19/1	196 & 208	32 33	30 (both)

   NOTE: Byte locations are given in decimal; bytes are counted starting from 0, not 1.

4. The loader is designed to select and load a series of programs. Since the disk only contains the VIP TERMINAL, press return when the "DESKTOP" screen is displayed. The program will load in about 2 minutes.

5. YOU'RE DONE!

UN-WRITE PROTECTOR

```
10 OPEN 15,8,15,"I0"
20 OPEN 2,8,2,"#"
30 PRINT#15, "U1:2 0 18 0"
40 PRINT#15, "B-P:2 2"
50 PRINT#2, "A";
60 PRINT#15, "M-W" CHR$(1) CHR$(1) CHR$(1) "A"
70 PRINT#15, "U2:2 0 18 0"
80 CLOSE 2
90 CLOSE 15
```

The program is MONTEZUMA'S REVENGE (R), Copyright 1984, Parker Brothers.

TYPE OF PROTECTION: The disk is filled from TRACKS 1 to 13 and TRACKS 19 through 36 with ERROR 21'S. This results in three "HEAD BANGINGS".

HOW TO COPY: Copy with any good nibble copy program that will place the errors for you.

HOW TO MAKE A WORKING COPY WITHOUT ERRORS:

1. An expander board is required for this program. A Cardco/5, Cartridge Backer or CSM Single Slot board will work. Power off the computer, insert the expander board and turn on the switch that grounds the EXROM line. This is switch 2 on the CB and CSM boards, and is the upper-right hand switch on the CARDCO/5. Power on the computer.

2. Load and execute HIMON with SYS 49152. Provide a clean work space with F 8000 BFFF 99. With this accomplished, perform a RESET with G FCE2. The screen will now display 30719 BYTES FREE.

3. Load the program from the original disk with LOAD ":*",8,1. When the program has finished loading and the game begins executing, perform a RESET with your reset switch. Re-enter the monitor with SYS 49152. With the M command, change the $37 at $0001 to a $36. This will flip out the BASIC ROM.

4. Using the M command, change location $8000 to $09. This will replace the $55, which was inserted as a result of the RESET routine. Refer to the PROGRAM PROTECTION MANUAL VOLUME II for a complete explanation of RESETS.

5. Save out the code from $8000 to $BFCC to a formatted disk with S "PBGAME",08,8000,BFCD.

6. To use this program LOAD "PBGAME",8,1. Activate with SYS 32777. This is the decimal equivalent for HEX $8009, which is the warm start vector for this program.

7. YOU'RE DONE!

The program is OXFORD PASCAL, Copyright 1984 O.C.S.S & D. Goodman.

TYPE OF PROTECTION: The disk contains error 23's on track 1, sector 0 and track 13, sector 9. Only the error at 13/9 is checked.

HOW TO COPY: Use any copy program which can copy error 23's.

HOW TO MAKE A WORKING COPY WITHOUT ERRORS ON THE DISK:

1) Copy the disk with any copy program which doesn't create errors.

2) Load a track/sector editor and run it.

3) Read in sector 18/1. Bytes 64-95 ($40-5F) should be all $00's now.
   Change bytes 66-72 ($42-48) to $82 0D 05 42 4F 4F 54.
   Change bytes 73-84 ($49-54) to $A0's (shifted spaces).
   Change byte 94 ($5E) to $0E.
   Write the altered sector back to disk,
   Note: the bytes in the sector are numbered starting at 0, not 1.

4) Read in sector 13/5.
   Change byte 4 ($04) from $04 to $08.
   Change byte 131 ($83) from $32 to $30.
   Change byte 138 ($8A) from $33 to $30.
   Write the altered sector back to disk.

5) Exit from the T/S editor. List the disk. The third file should be a new program 14 blocks long called BOOT.

6) Scratch the files PASCAL and LD from the disk. Validate the disk using the DOS command "V0". List the disk. You should have 134 blocks free.

7) You're done!

The first program on the disk, PASCAL, is a BASIC program which just loads in the file LD. LD is an autoboot routine (using the stack) which loads at $0100-$18FF and starts executing at $1770. LD then loads 14 blocks from disk into the BASIC area at $0801-1521. This 14-block program starts at sector 13/5 and is linked together like a regular program file. In step 3 above we created a directory entry for this program (in a conveniently empty slot) and named it BOOT. Refer to PPM Vol I for details on how directory entries are set up.

Let's take a look at how LD can load in the BOOT program without a directory entry. Remember, each sector of a normal file starts off with a two-byte link to the next sector of the file. The normal LOAD process gets the first link from the program's directory entry, and simply follows the links from there. LD starts by disabling the NMI vector (RESTORE key), initializing the disk (I0) and opening a random access channel (#). Then it initializes some key locations: $45-46 holds the address to begin loading at ($0801), $47-48 holds the first link ($000D 05 = 13/5) and $49 holds the number of bytes to read from each sector, not counting the link bytes ($FE). Next, the main routine calls a subroutine at $182E-64. That routine calls on another routine at $1865-8A, which converts a hex byte into its two-byte ASCII equivalent and puts the result into $62-63. In this case, the routine is called twice to convert the T/S link to ASCII, since a U1 command requires its T/S info in ASCII. The ASCII T/S info is then written into a 'blank' U1 command at $17AB-B6. The U1 command is sent to the drive, which gets the specified block. After any U1 command, the disk sends an error (or OK) message, so a routine at $18CA-DC is called to read in the message without doing anything with it (just to 'shut the drive up').

Finally, we return to the main routine with the first block ready to be read into computer memory. The first two bytes contain the next T/S link, so a subroutine at $18ED-FA is called which simply opens up the data channel (2) and reads in one byte. This byte (track #) is stored at $47 and then the routine is called again to get the next byte (sector #), which is stored at $48. This saves the T/S link for the next U1 command. Next, a routine at $1825-2D is called which checks to see if this is the last sector of the file. Since it isn't, we'll postpone looking at that routine until a little later. In fact, this is the very first block of the file, so the next two bytes are the load address. The main routine is going to load the file into memory based on its own pointer at $45-46 (set to $0801 earlier), so the load address bytes are simply read in by the $18ED routine and ignored. Location $000D will be used to count the number of bytes read from the sector so far (not including link bytes), so this counter is set to $02 to account for the load address bytes.

At long last we are ready to read in the actual program code from the first sector. The routine at $18ED is used to get the bytes one at a time. The subroutine at $18E2 stores each byte in the location pointed to by the pointer at $45-46, which is then incremented. Back in the main routine the byte counter in $000D is incremented too. Once the number of bytes specified by $49 has been read in, we are ready to get the next sector, assuming there is one. The routine at $182E is called to convert the T/S link to ASCII, send out the U1 command and skip over the error (OK) message. Then the first two bytes of the new sector (next T/S link) are saved in $47-48 and checked to see if this is the last sector. Since it isn't the last sector, we simply set the number of bytes read so far ($49) to $00 and jump back to the code which reads in the rest of the sector.

The process described in the last paragraph is repeated over and over until the last sector is encountered. The LD pogram detects this the same way the normal LOAD routine does: the first byte of the T/S link is $00 (there is no track 0). The second byte of the link then specifies where in this block the file ends (since the block may not be full). The routine at $1825 transfers this byte to location $49 (number of bytes to be read). Then the rest of the block is read in, up to the specified byte. The error (OK) message is skipped, and the files are closed. The real BOOT program has now been read in and is ready to be executed.

The BOOT program is executed with JMP $080F (=2063). The author has conveniently preceded the code at $080F with a BASIC program consisting of SYS (2063), which means we can start the program with RUN. The program checks sector 13/9 for an error 23 in a completely straightforward manner. Step 4 above changes the check for error 23 into a check for no error (00) by changing a CMP #$32 and CMP #$33 to CMP #30's as usual. Also in step 4, we change the default load address of the BOOT program from $0401 to $0801. This is not absolutely necessary, but it avoids problems if you should accidentally load BOOT using ",8,1" rather than just ",8". Once we have put BOOT into the directory, we no longer need the PASCAL and LD files. We scratch them and then validate the disk to correct the BAM (which indicated 0 blocks free origfnally). The disk can now be file copied or have other programs added to it. To start PASCAL, simply load BOOT and RUN.

CUSTOMIZING THE KERNAL

Everybody has their own list of changes they'd like to make to their Commodore 64. Maybe you don't like the screen colors, or you're tired of typing ",8" every time you load a program, or you'd like the screen to say 'WELCOME TO BOB'S COMPUTER' when you turn the computer on. These and other changes can be accomplished easily. You'll need an EPROM programmer, a 2764 EPROM and a 24/28-pin adapter. A reference book such as 'The Anatomy of the Commodore 64' is also immensely helpful, especially if you want to go beyond the modifications we present here.

To start off, you should get a copy of revision 3 of the KERNAL if you don't already have it (SX-64 owners can use their present KERNAL). You could proceed by copying the KERNAL down to RAM memory at, say, $4000-5FFF, but then you'll have to translate addresses back and forth. A better way is to copy the KERNAL ROM down to the RAM underneath it and switch out the ROM. You can run your system from the RAM copy as long as you don't try to modify a system routine as it is being executed (such as the screen editor and IRQ routines). This way you'll be able to test your changes as you make them. You'll have to copy the BASIC ROM down to RAM as well, since BASIC gets switched off with the KERNAL. You won't be able to use a cartridge based monitor either, since cartridges are also switched out. Of course, you should save your custom KERNAL to disk from time to time as a precaution, but this method works well for us.

Start by copying BASIC and the KERNAL down to RAM with T A000 BFFF A000 (BASIC) and T E000 FFFF E000 (KERNAL). Now switch off the ROMs by changing location $0001 to a $35. You shouldn't notice any change, but you're now operating out of RAM. Now change location $FDD6 to $E5 (be sure and change it back to $E7 before burning an EPROM copy!). This change prevents the RESET routine from switching the ROMs back in when you do a 'soft' RESET (SYS 64738 or G FCE2). Since many modifications will change the startup defaults, you'll need a soft RESET to test them. A 'hard' RESET with a RESET button will still switch the ROMs back in, but it won't wipe out your RAM copy. Try a soft RESET now with G FCE2. After an extra-long pause you will get the familiar startup screen, only with 51216 bytes free! Relax, this just means the RAM test routine got all the way to the VIC control register at $D011 before finding any 'non-RAM'.

The table on the next page lists some short patches you can make, just for starters. Let's take a brief look at each of them, starting with everyone's favorite, the default colors. The default border and background colors are located in a table of values that get written into the VIC chip on powerup, RESET or RUN/STOP-RESTORE by the VIC initialization routine at $E5AA. The border and background color codes are located at ECD9-DA, respectively. These colors are specified using the color RAM codes ($00=black, $01=white, etc.) NOT the ASCII print commands ($90=black etc.). The default character color is located at $E535, in the screen initialization routine (which also calls the VIC chip init routine). The character color is also specified using color RAM codes.

Next on our list is a patch to automatically enable repeating for all keys. To do this we'll have to insert a JSR in the screen init routine and put a short piece of code in the unused area at $E4B7-D2. This simply stores an $80 into the key repeat flag and returns. While you're at it, you might want to adjust the delay before a key repeats and the speed at which it repeats (note two different locations to change speed). The new values given are just suggestions, so experiment.

Now we come to a real handy pair of changes. First, we modify the BASIC LOAD and SAVE commands so they automatically go to disk rather than tape when no device is specified (i.e. LOAD "PROG"). If you need to specify a secondary address, though, you'll still have to type the device number (LOAD "PROG",8,1). Likewise, you can default the OPEN command to go to the printer if no device is specified. You'll still have to supply the device number if you need a secondary address. Neither of these changes affect the corresponding KERNAL routines, since you must always specify the device number for them (unless you want to get really tricky).

Our next modification is to change the SHIFT-RUN/STOP (Sh-R/S) combination so that it loads the first program off the disk and runs it. To do this, you must first change the default LOAD device as given above. The Sh-R/S combination works by putting its command into the keyboard buffer and then jumping to the keyboard interpret loop. Since the keyboard buffer is only 10 characters long, this limits our possible modifications. To fit in a LOAD ":*" (return) RUN (return) command, we have to use the shifted shortcuts, e.g. R shift-U for RUN, shown as rU. The only way to get around this limitation is to patch into the keyboard interpret loop at $E5EE, and have it perform your custom command directly.

Our final modification involves the startup screen, which is printed in three separate parts by a routine at $E422-46. The first part, up to '64K RAM SYSTEM ', is stored as a table of ASCII characters at $E473-AB. The number of bytes free is calculated and printed next, and then the 'BASIC BYTES FREE' message at $E45F-72 is printed. The first part printed can be altered by just changing the ASCII codes. To disable the bytes free and reuse its message area, make the last two modifications shown. You can then continue your screen message in the 'BASIC BYTES FREE' area.

If you want to go much beyond the modifications here, you'll probably need some extra space to put code in. The only unused area left is at $E4B7-D2, but you could free up some space (nearly 2K!) if you don't need the cassette. Once you've made your changes, burn the code into a 2764 EPROM and install it in the computer using the AD adapter. Powerup and take your new computer for a test drive!

LOCATION	ORIGINAL	MODIFIED	DESCRIPTION OF MODIFICATION
FDD6	E7	E5 see text	CHANGE TO $E7 BEFORE BURNNG EPROM!
ECD9-DA	0E 06	Your choice	Default border, background colors
E535	0E	Your choice	Default character color
E536-8 E4B7-BF	STA $0286 AA ... . AA (filler)	JSR $E4B7 STA $0286 LDA #$80 STA $028A RTS	Patch to key repeat routine Save character color Load value - repeat all keys Save in key repeat flag Return to screen init routine
E53A/EB1D EAEA	04 10	03 08	Key repeat speed - change 2 places Key repeat delay
E1DA	01 (tape)	08 (disk)	Change LOAD/SAVE default device
E228	01 (tape)	04 (printer)	Change OPEN default device
E5EF E5F4-F5 E4C0-C9	09 E6 EC AA ... AA	0A C0 E4 4C CF 22 3A 2A 22 0D 52 D5 0D	No. chars in SHIFT-RUN/STOP command Location of " " " " New version 10":*" (return) rU (return)
E473-AB E45F-72 E430-32	"***** COM.." "BYTES FREE" LDA $37; SEC	Your choice Your choice JMP $E43D	Change startup message (see text) Reuse bytes free space Disable no. bytes free calc/print

EXECUTING 1541 ROUTINES

As program protection evolves, schemes involving bad blocks checked with a U1 command are becoming less common (finally). The newer schemes often use custom DOS routines executed inside the disk drive's memory. There are several ways to accomplish this, including MEMORY-EXECUTE (M-E), BLOCK-EXECUTE (B-E), ampersand files (&:) and the job queue commands JUMP ($D0) and EXECUTE ($E0). Ampersand files are very rarely used, so we won't consider them at this time. We'll start our discussion with M-E.

M-E is used to execute a routine which is already in the drive's memory. The routine can either be in the drive's 2K of RAM or in its 16K of ROM. The only requirements are that you know where the routine is located and that the routine end with an RTS. Books such as THE ANATOMY OF THE 1541 DISK DRIVE and INSIDE COMMODORE DOS will help you to find useful routines in the DOS ROMs. In the case of a RAM routine, you have to put the routine in memory yourself before you can execute it. There are a couple of different ways you might do this.

The first way is to use a MEMORY-WRITE (M-W) command. This command can be used from either BASIC or machine language. You must open the command channel first (e.g. OPEN 15,8,15) and then send the M-W command through this channel. The general form of the M-W command in BASIC is:

```
PRINT#15,"M-W";CHR$(lo);CHR$(hi);CHR$(n);CHR$(d1);CHR$(d2); ...
```

'lo' and 'hi' specify the low and high bytes of the address in the drive's memory to start putting the data in, 'n' is the number of bytes you will send in this M-W command (up to a maximum of 34) and 'd1', 'd2' etc. are the data bytes themselves. You must send all this information (except the "M-W" itself) in CHR$() form, not ASCII. For instance, if you want to send 5 bytes, you must specify this with CHR$(5), NOT "5" (which is the same as CHR$(53)). Also, you should use semicolons (;) as punctuation in BASIC, not commas. The drive expects each byte of information to be in a fixed position in the command and the extra spaces inserted by a comma will throw it off. Most important of all, DO NOT put a colon (:) after the "M-W" or use "MEMORY-WRITE" instead of "M-W". The older disk manuals and many reference books show these being used but they don't work. This has been corrected in the newest Commodore manuals.

Another way to get a routine into 1541 RAM is to load it directly from diskette using either the BLOCK-READ (B-R) or USER1 (U1) commands. The U1 command is by far the most common way, since B-R has several bugs in it. When using U1, special care must be taken to ensure that the block is loaded into the correct memory location. You must first open the command channel to the disk and then open up a # channel too. A typical example of opening a # channel is :

```
OPEN 1,8,2,"#"
```

This form lets the disk drive select which of its buffers to load the block into. The drive has five 256-byte buffers available, from buffer 0 at $0300-03FF on up to buffer 4 at $0700-07FF. Since many ML routines must reside in a particular place in memory, and since you have to know where a routine is to execute it with M-E, you need to be able to select a specific buffer to be used. Fortunately, you can control this by simply specifying the buffer number after the "#", e.g. OPEN 1,8,2,"#0". Having opened the # channel, you are ready to send the U1 command. The syntax for U1 is :

```
PRINT#15,"U1:";channel;drive;track;sector
```

'Channel' is the channel specified in the secondary address of the OPEN command (2 in the above example), NOT the buffer number (0). 'Drive' must be 0 on the 1541. In contrast to the M-W (and M-E) commands, the information in a U1 command must be sent as ASCII characters. This can be done several ways. For example, a 5 could be sent as using the string "5", the number 5 or the variable V (assuming V = 5). It could even be sent as CHR$(53), the ASCII value of "5", BUT NOT AS CHR$(5). You may use a colon after the U1 but it's not required. You may use a space or almost any character as long as there is SOMETHING there. For punctuation you should use semicolons, although spaces are acceptable within strings. The following are valid U1 commands:

```
PRINT#15,"U1:";2;0;1;0
PRINT#15,"U1:2 0 1 0"
```

Once the routine has been read in with U1, sent via M-W or located in ROM, you are ready to execute it with an M-E command. This is actually the simplest part of the process. The syntax is:

```
PRINT#15,"M-E";CHR$(lo);CHR$(hi)
```

'Lo' and 'hi' specify the address to start executing at. The M-E command follows the same syntax rules as M-W. Use CHR$() not ASCII; use semicolons, not commas; and **DON'T PUT A COLON AFTER THE M-E.**

This brings us to the BLOCK-EXECUTE (B-E) command, which is a combination of U1 and M-E. B-E loads a block from disk and executes it automatically. B-E requires a # channel like U1 and follows the same syntax too. Since B-E always starts executing at the beginning of the buffer, you don't have to specify an execution address. With M-E you can start anywhere in the buffer you wish. This is the only difference between using B-E and using U1/M-E.

There are two other alternatives to M-E: the job queue commands JUMP ($D0) and EXECUTE ($E0). Job queue commands are activated by putting their hex codes directly into the job queue located at $0000-0005 in the 1541 (using M-W, for instance). This area is examined by the 1541's processor at every IRQ interrupt. If a valid job code is found, that task will be performed. Each job in the queue has a corresponding buffer and track/sector pointer. Job 0 (location $0000) uses buffer 0 ($0300-03FF) and gets its T/S info from location $0006-7. Job 1 uses buffer 1 and location $0008-9, and so on (job 5 has no buffer). Routines executed through the job queue should end by storing a status code back over their own job code in the queue and then jumping to the DOS's idle loop (wait for something to do). If the job code isn't wiped out, the routine will be re-executed over and over.

The JUMP ($D0) command simply jumps to the beginning of the corresponding buffer and starts executing. The EXECUTE ($E0) command is a deluxe command which positions the read/write head to the track specified in its T/S pointer, selects the proper density and waits for the drive motor to come up to speed before executing the routine in its buffer. This makes EXECUTE very convenient for program protection routines which check the disk for some type of custom formatting. If you use M-E or JUMP, you must handle all this yourself. Of course, many times you'll want to change the density, move to a half-track, etc. anyway. What you do is up to you; the M-E, B-E, JUMP and EXECUTE commands are your keys to controlling the disk drive.

MAYBE IT WILL WORK DEPARTMENT (MAYBE NOT)

HOW DO YOU WRITE A TRACK ADJACENT TO A HALF/TRACK?

On the 1541 it is very diffilcult (if not impossible). This is due to the write with erase feature of the drive that erases a guard band between tracks. When you write a track, the adjacent halftrack will be erased by the guard band. Unless.... you want to try modifying your drive. This is a procedure that sometimes works (sometimes it don't). If the track and the half-track contain identical information (as does most of ARTs disks) this method may help. NOTICE: ANY MODIFICATION TO THE DISK DRIVE MAY RESULT IN DAMAGE AND/OR INJURY. CSM SOFTWARE INC. WILL NOT ACCEPT ANY RESPONIBILITY FOR DAMAGE AND/OR INJURY AS A RESULT OF THIS PROCEDURE OR ANY OTHER PROCEDURE!

1. Locate the connector to the read write head on your disk drive. This is a black plastic connector that uses wires 1, 2, 3, and 5 (4 is not used).

2. Remove the #3 wire (normally the yellow wire) from the connector. This wire supplies the current to the ERASE portion of the R/W head, thereby preventing the guard band from being erased.

If your R/W head produces a wide enough track it just may be interperted by the drive as having the same data on the track and the half-track.

Originally these disks that contain data on the track and the adjacent half-tracks may have been created on drive with a 96 track per inch (tpi) R/W head assembly. The 1541 uses a 40 tpi R/W head assembly. As you can imagine, the drive that uses the 96 tpi R/W head will produce a much narrower track than one produced with a 40 tpi head. If the actual track is narrower the guard band will be narrower. With this information in mind you can see how it is possible to write adjacent tracks and half-tracks. With our little hardware modification it is possible to force the disk drive into writing a little track wider than normal. This extra wide track may be interpreted by the drive as data on the track and on the half-track. Just keep in mind that this procedure does not always work as outlined. This is undoubtly due to variations of the drive and of the disk media.

3. Be sure to reinstall the wire after testing the procedure. AS WITH ALL HARDWARE MODIFICATIONS... USE CAUTION!!

NEW FROM CSM

CSM now offers a NUMERIC KEYPAD for the C-64 and VIC-20. Our keypad has more features than any other on the market. It plugs into the keyboard connector inside your computer rather than the joystick port, so it is 100% compatible with all software. In addition to the numeric keys and decimal point, it features SHIFT, RETURN, INST/DEL and cursor keys. It also has keys for all the arithmetic operations including + - * / and ↑.

P.S. If you remember, Bear Technology's keypad, this is the same fine product.

JUNE 1985

ALL ITEMS IN STOCK FOR IMMEDIATE SHIPMENT

PROGRAM PROTECTION MANUAL FOR THE C-64 (VOLUME II) $34.95
THIS MANUAL BEGINS WHERE THE FIRST ENDED. COVERS THE LATEST ADVANCES IN PROGRAM PROTECTION INCLUDING ENCRYPTION, GCR CODING, UNDOCUMENTED OPCODES, ADVANCED CARTRIDGE PROTECTION SCHEMES, AND MUCH MORE.

1541 DISK DRIVE ALIGNMENT PROGRAM - NEW!! VERSION 2.0 44.95
See Oct. 84 Compute! Gazette for review

PROGRAM PROTECTION MANUAL FOR THE C-64 (VOLUME I) 29.95

CARDO 5 SLOT EXPANSION BOARD FOR THE C-64 65.00
INCLUDES A RESET BUTTON AND FULL CARTRIDGE SWITCHES

ADVANCED ML BOOK FOR THE C-64	(ABACUS)	19.95
ANATOMY OF THE 64	"	19.95
ANATOMY OF THE 1541	"	19,95
MACHINE LANGUAGE BOOK OF THE 64	"	14.95
TRICKS AND TIPS FOR THE 64	"	19.95
1541 MAINTENANCE MANUAL	(GOSUB)	29.95
INSIDE COMMODORE DOS	(DATAMOST)	19.95

PROMENADE 99.50
THIS IS THE EPROM PROGRAMMER THAT WE USE HERE!! USE IT TO MAKE YOUR OWN OPERATING SYSTEM CHIPS, CARTRIDGE PROGRAMS OR A NEW DOS FOR THE DISK DRIVE. SOFTWARE INCLUDED ON DISKETTE.

DATARASE This will erase all U.V. erasable EPROMS (2732, 2764, 27128, 27256, etc.) 34.95

** NEW LOW PRICE!

BOARDS, SOCKETS AND EPROMS:

PCC2	2 CHIP CARTRIDGE BOARD WITH 2 SOCKETS	** 5.95
PCC4	4 CHIP BANK SWITCHED CARTRIDGE BOARD W/SOCKETS	19.95
PCVH2	PLASTIC HOUSING FOR ABOVE BOARDS	** 2.50
LP28	28 PIN SOCKET FOR EPROM	.59
ZIF28	ZERO INSERTION FORCE 28 PIN SOCKET	4.75
2732	EPROM (USE IN CARTRIDGE 4K)	5.00
2764	EPROM (USE IN CARTRIDGE 8K OR IN DISK DRIVE W/AD)	** 6.00
27128	EPROM (16 K ON ONE CHIP - USE IN PCC4)	** 14.00
27256	EPROM (32 K ON ONE CHIP - USE IN PCC4)	** 18.00
AD	USE THIS ADAPTER WITH THE 2764 EPROM IN THE DISK DRIVE OR IN THE COMPUTER.	** 6.95

CARTRIDGE PACK - NEW!! ** 17.50
INCLUDES A PCC2 2-CHIP BOARD WITH 2 SOCKETS INSTALLED, TWO 2764 EPROMS AND A PCVH2 PLASTIC CASE.

HESMON (R) ML MONITOR ON A CARTRIDGE. 25.00
This version also includes a DOS WEDGE and PROMOS (PROMENADE) software on the cartridge.

SINGLE SLOT EXPANSION BOARD 24.50
WITH RESET BUTTON AND GAME, EXROM, ENABLE & POWER SWITCHES.

XETEC SPI PRINTER INTERFACE 54.95

DISK DRIVE BELT 10.00

1541 DISK DRIVE ALIGNMENT PROGRAM - VERSION 2.0 44.95

NO SPECIAL TOOLS OR KNOWLEDGE REQUIRED, JUST AVERAGE MECHANICAL ABILITY.

VERSION 2.0 IS NOW AVAILABLE. THE BEST JUST GOT BETTER! VERSION 2.0 FEATURES AN EVEN MORE ACCURATE CALIBRATION DISK, PREPARED ON SPECIAL EQUIPMENT JUST FOR CSM.

A NEWLY REVISED MANUAL WITH TROUBLESHOOTING TIPS IS INCLUDED.

UPDATES ARE AVAILABLE FOR REGISTERED PRIOR PURCHASERS - 20.00

* * *

CARTRIDGE BACKER 54.95

THIS WILL 'BACK UP' 99% OF ALL CARTRIDGES ON THE MARKET TODAY! IT WILL DOWN LOAD THE NEW VERSION OF THE PROGRAM TO DISK AND WRITE AN AUTO BOOT LOADER FOR IT. AS MANY AS TEN CARTRIDGES MAY BE STORED ON A DISK. IT WILL ALSO TELL YOU WHAT CODE IT CHANGED IN ORDER TO 'BACK THEM UP'. THE UNIT CONTAINS IMPROVED SOFTWARE THAT WILL ALLOW YOU BACK UP ALMOST EVERY CARTRIDGE AVAILABLE. IT WILL ALSO BACK UP MANY DISK PROGRAMS (ARTS, ACTIVE, PARKERS AND MORE). THE PROGRAM ALSO CONTAINS SUPER SAVER - DISK TO DISK AND DISK TO TAPE FILE COPYING.

CARTRIDGE BACKER does not require the use of an 'EXTERNAL RAM BOARD' in order to run the disk version of the program, as do some of the other cartridge backup utilities on the market.

* * *

PROGRAM PROTECTION MANUAL - VOLUME II 34.95

THIS MANUAL STARTS WHERE THE FIRST LEFT OFF... A FULL INTRODUCTION TO MACHINE LANGUAGE, RESETS, INTERRUPTS AND AUTO BOOTS. IT ALSO COVERS THE MOST RECENT ADVANCES IN PROGRAM PROTECTION. HALF-TRACKS, EXTRA TRACKS AND SECTORS, NON-STANDARD FORMATS, NIBBLE COUNTING, TRACK ARCING, UNIMPLEMENTED OPCODES, ENCRYPTION TECHNIQUES AND MUCH, MUCH MORE!!! EXPERTS HAVE AGREED TO CONTRIBUTE SELECTED CHAPTERS IN THEIR OWN FIELD OF EXPERTISE. THE INFORMATION IS PRESENTED IN AN EASY-TO-UNDERSTAND FASHION FOR THE BEGINNERS AND EXPERTS ALIKE. SAMPLE PROGRAMS ARE INCLUDED TO HELP YOU UNDERSTAND ALL THE PRINCIPLES.

* * *

CSM SOFTWARE INC.
P. O. BOX 563
CROWN POINT, IN 46307
(219) 663-4335

all items plus shipping - $3.50 for most items